A fast solution method for the time-dependent orienteering problem

C. Verbeeck[a,*], K. Sörensen[b], E.-H. Aghezzaf[a], P. Vansteenwegen[a,c]

[a] *Ghent University, Department of Industrial management*
[b] *University of Antwerp Operations Research Group ANT/OR*
[c] *KU Leuven, Centre of Industrial Management, Traffic and Infrastructure*

Abstract

This paper introduces a fast solution procedure to solve 100-node instances of the time-dependent orienteering problem (TD-OP) within a few seconds of computation time. Orienteering problems occur in logistic situations were an optimal combination of locations needs to be selected and the routing between the selected locations needs to be optimized. In the time-dependent variant, the travel time between two locations depends on the departure time at the first location. Next to a mathematical formulation of the TD-OP, the main contribution of this paper is the design of a fast and effective algorithm to tackle this problem. This algorithm combines the principles of an ant colony system (ACS) with a time-dependent local search procedure equipped with a local evaluation metric. Additionally, realistic benchmark instances with varying size and properties are constructed. The average score gap with the known optimal solution on these test instances is only 1.4% with an average computation time of 0.5 seconds. An extensive sensitivity analysis shows that the performance of the algorithm is insensitive to small changes in its parameter settings.

Keywords: Metaheuristics, Vehicle Routing, Orienteering problem

1. Introduction

The orienteering problem (OP) is defined on a graph in which scores are assigned to the vertices and a travel time is assigned to each edge linking two vertices. The objective of the OP is to select a subset of vertices and determine the order in which they are visited so that the total collected score is maximized while the maximum total travel time is not exceeded. In addition, a feasible OP solution should start and end at a predetermined vertex. The OP integrates the knapsack problem (KP) and the traveling salesperson problem (TSP). In contrast to the TSP, not all vertices can be visited in an OP due to the maximum travel time constraint. However, determining the shortest path visiting the selected vertices helps to visit more vertices and might increase the collected score.

OPs are typically used in logistic planning tools where each vertex represents a customer and the score reflects the profit margin achieved by visiting this customer. The aim of a logistic company is to select the combination and sequence of customers that maximizes the total profit [42, 17, 21]. Furthermore, OPs serve as the basic problem formulation for personalized touristic trip planners [37, 47, 38, 46]. In this case, each vertex is a point of interest (POI) and the score of a POI indicates the personal interest that the tourist attaches to it. Since a tourist generally does not have the time to visit all possible POIs, personalized trip planners can use an OP to propose the highest scoring combination of POIs that can be visited within the time limit set forth by the tourist. For a longer list of practical and real-life applications of the OP and its variants, we refer to the recent survey by Vansteenwegen et al. [45].

This research focuses on *time-dependent orienteering problems* (TD-OP) in which the travel time between two vertices depends on the departure time at the first vertex. This specific problem formulation allows to tackle congestion related issues in routing problems such as morning and evening peaks on the highways or crowded city center traffic situations. Also, multi-modal applications for logistic or touristic trip planners rely on TD-OP solution methods. The

*Corresponding author: Ghent University, Technologiepark 903, 9052 Zwijnaarde, Belgium, Tel.: +32 9 264 54 95, Fax.:+32 9 264 58 47
Email addresses: cedric.verbeeck@ugent.be (C. Verbeeck), kenneth.sorensen@ua.ac.be (K. Sörensen), elhoussaine.aghezzaf@ugent.be (E.-H. Aghezzaf), pieter.vansteenwegen@cib.kuleuven.be (P. Vansteenwegen)

most common example is the combination of walking and using public transport, where the waiting time at a bus station and the time table of the bus result in time-dependent travel times [14]. In general, we can state that taking into account time-dependent travel times in routing problems makes them more realistic. Furthermore, due to the rise in congestion problems on the one hand and the acceptance of smartphones and PDA's with GPS and internet connection on the other, the construction and update of routes, based on new congestion information increasingly becomes a necessity for a number of business applications. More importantly this construction and update needs to be done in a very short time span [26] and therefore fast algorithms are required to update previously scheduled routes when new traffic information becomes available. To conclude: taking into account time-dependent travel times becomes necessary for an implementation of the algorithms in practice. Moreover, the time-dependent vehicle routing problem, a related problem, has received a lot of attention lately [19, 9, 18, 35, 7, 43, 30, 25]. For all these reasons, the addition of time-dependent travel times to the basic OP is an obvious step in the direction of modeling and solving realistic routing problems.

Following a literature review in Section 2, the TD-OP is defined mathematically in Section 3. Then, a local-search based metaheuristic is proposed in Section 4, and experimentally tested in Section 5. Section 6 concludes this paper and introduces possible future work.

2. Literature review

The name of the orienteering problem derives from the sport game of orienteering [42, 6]. In this game, individual competitors start at a specified control point and try to maximize their score by visiting checkpoints and returning to the control point within a given time frame. Each checkpoint has a known score and the objective is to maximize the total collected score. The OP is also known as the selective traveling salesperson problem [28, 15, 41], the maximum collection problem [22, 5] and the bank robber problem [3]. Well-known extensions to the OP are the team OP (TOP), and the OP with time windows (OPTW). The first extension allows to plan trips for multiple days or to make use of multiple vehicles. In the second extension, a time period is defined for each vertex in which it can be visited. The team orienteering problem with time windows (TOPTW) combines both extensions. A recent survey on the OP and its extensions can be found in Vansteenwegen et al. [45]. Since the OP is NP-hard [17], exact algorithms are time-consuming and most research is therefore focused on heuristic approaches such as the ones found in [42, 17, 36, 6, 16, 38].

The time-dependent variant of the OP is relatively new and has, to the best of our knowledge, only been studied by [13, 1, 14, 32, 31]. Fomin & Lingas [13] were the first authors to mention the TD-OP and state that it is NP-hard because the OP is NP-hard. However, they do not develop an algorithm for the TD-OP that can be used in practical situations. Abbaspour & Samadzadegan [1] introduce a solution procedure for the TD-OP with time windows based on two adaptive genetic algorithms and multi-modal shortest path finding modules. They are able to solve multi-modal routing problems in the city of Tehran, although no absolute performance measure (gap) was reported. Li et al. [32] propose a mixed integer programming model of the TD-OP combined with an optimal pre-node labeling algorithm based on the idea of network planning and dynamic programming. However, this algorithm is not tested on test instances and therefore no performance metrics are proposed. The same conclusion holds for Li [31] where a mixed integer programming model is proposed and an optimal dynamic labeling algorithm is designed for the time dependent team orienteering problem (TD-TOP). Finally, Garcia et al. [14] develop an iterated local search heuristic for the time-dependent team orienteering problem with time windows (TD-TOPTW) which allows them to illustrate, based on a case study in the city of San Sebastian, that obtaining near-optimal routes in a few seconds is feasible. However, only a special case of time dependency is considered, which is the result of using public transport in a city environment. For example, when a traveler arrives at a bus stop before the bus arrives he needs to wait. Therefore the travel time between two locations consists of both the waiting time and the driving time and depends on the departing time at the start location. They exploit the fixed frequency of bus services to come up with an efficient solution technique.

Existing solution methods for the time-dependent vehicle routing problem (TD-VRP) [19, 9, 18, 35, 7, 43, 30, 27, 25, 8], a related problem, can provide inspiration on how to deal efficiently with time dependency. Tabu search is the most commonly applied metaheuristic for the TD-VRP [19, 43, 30]. Other algorithms that have been developed for the TD-VRP include a genetic algorithm [18], a heuristic combining route construction and route improvement [35, 7], a variable neighbourhood approach [27] and an ant colony system [9]. Kok et al. [25] use a time-dependent shortest

path algorithm together with a restricted dynamic programming heuristic to tackle real-life TD-VRP test instances. Dabia et al. [8] developed a branch-and-price algorithm for the time-dependent vehicle routing problem with time windows (TD-VRPTW). They used new dominance criteria which enabled them to solve some small and medium sized instances.

An issue that has received some attention in the time-dependent routing literature is the modeling of time-dependent travel times and the generation of realistic benchmark instances. One of the first approaches defines the travel time between nodes as a function of the distance and the time of day by using a time-dependent cost factor. This results in a piecewise constant distribution of the travel time [33, 34, 35]. Although this technique is easily applied, the generated travel times violate the FIFO principle, as shown by Ichoua et al. [19]. The FIFO principle states that if two vehicles leave from the same location to the same destination and travel on the same path, the one that leaves first also has to arrive first. The proposed speed models of [19, 9] are able to model more realistic congestion behavior between locations and satisfy the FIFO principle. By adopting such a speed model more realistic benchmark instances can be created by working with step-like speed distributions and adjusting the travel speed whenever the boundary between two consecutive time buckets is crossed. An important decision concerns the way the travel time data is stored. Donati et al. [9] divide the planning period (e.g., one day) in fixed-width time buckets (e.g., one hour). Ichoua et al. [19] adjust the bucket width to the expected travel time profile, still using the same bucket boundaries for all arcs. The latter method is more realistic as the travel time fluctuations due to rush hours are better modeled. Still, these rush-hours do not necessarily occur at the same time of the day on all the arcs. On the other hand, dividing the travel time into time buckets that are different for each arc, turns out to be quite difficult as these matrices have to be created artificially. This approach has been used by Chen et al. [7] and Malandraki & Daskin [34] who arbitrarily create variable time zones for all edges but without mentioning a procedure to control for *spatial* consistency.

In order to realistically imitate real-life traffic congestion, it is not enough to be time consistent (FIFO-conforming). The speed model should also be spatial consistent as congestion tends to grow and shrink in spatially correlated zones and not independently on individual roads. Both time and spatial consistency as defined by Lecluyse et al. [29], will be taken into account in this paper when new benchmark instances for the TD-OP are created.

3. Problem description

3.1. A time-dependent travel speed model

In time-dependent routing problems, a so-called *speed model* can be used to determine the travel time between two vertices on a specific moment in time. In this section, the speed model used in this paper, together with the necessary input data and assumptions, are discussed. This speed model for the TD-OP is based on the speed model of Ichoua et al. [19] and Donati et al. [9] for the TD-VRP. However, different arc categories and different time buckets (time periods) with non-equal width were used. More importantly, during the construction of the problem instances the arcs are not randomly assigned to an arc category.

In our speed model, the speed, and therefore the travel time, of a vehicle on an arc depends on the time periods it is traveling in and the arc category. Four unequal time periods, reflecting a congestion peak in the morning and in the evening, alternated by a period with normal traffic conditions, are incorporated into the speed model. In addition to this, five arc categories are also included.

- *Always busy*: these arcs represent busy city centers with a lot of traffic during the whole day
- *Morning peak*: these arcs represent roads leading from a living area to the city center, typically congested in the morning
- *Two peaks a day*: these arcs represent roads near the highway with a morning and evening peak in both directions
- *Evening peak*: these arcs represent roads leading from a city center to a living area, typically congested in the evening
- *Seldom traveled*: these arcs represent roads in rural and less traveled areas

A speed $v_{c,k}$ is defined for every combination of time period k, $1 \leq k \leq K$ and arc category c, $1 \leq c \leq C$. The used speed matrix with $K = 4$ and $C = 5$ is displayed in Table 1. Note that the average speed in the displayed speed matrix is equal to 1, resembling the speed in the time-*independent* problem. Furthermore, we assume that the time periods are the same for every arc category. This speed model adheres to the FIFO principle. That is, leaving a node earlier

guarantees that one will arrive earlier at the destination. Furthermore, since we only want to construct routes during the daytime (cfr. 3.3), we assume that trucks leave the depot after 7 am and return before 9 pm.

Table 1: Used speed matrix

Congestion description Time periods ($\underline{tp}_k$–$\overline{tp}_k$)	Morning peak 7am–9am	Normal 9am–5pm	Evening peak 5pm–7pm	Normal 7pm–9pm
Arc categories (c)				
1. Always busy	0.5	0.81	0.5	0.81
2. Morning peak	0.5	0.7	1	1.5
3. Two peaks	0.5	1.5	0.5	1.5
4. Evening peak	1	1.5	0.5	0.7
5. Seldom traveled	1.5	1.5	1.5	1.5

To calculate the travel time between two vertices (i and j), we need the distance $d_{i,j}$ between these vertices and the departure time t_d. The departure time defines the starting time period and together with the arc category we can look up the starting speed $v_{c,k}$ in the speed matrix. When the vehicle travels, the end of its starting time period can be reached. At that moment a new time period, together with a new speed, will be entered. This is repeated until the vehicle arrives at vertex j. The travel time between two vertices can be calculated by adding the sub-travel times in every time period until vertex j has been reached. This procedure has been explained in [19] and our corresponding pseudo code is listed below in Algorithm 1. Let $\underline{tp}_k$ and $\overline{tp}_k$ represent the moment at which time period k, starts and ends respectively. We suppose that the vehicle leaves vertex i at the departure time $t_d \in [\underline{tp}_k, \overline{tp}_k[$ and that the arc (i, j) belongs to category c. Next, t denotes the current time, t_a denotes the arrival time and t_d denotes the departure time.

Algorithm 1 travel time calculation - input: i, j, t_d

$t \leftarrow t_d$
$d \leftarrow d_{i,j}$
find k such that $t_d \in [\underline{tp}_k, \overline{tp}_k[$
$t_a \leftarrow t + (d/v_{c,k})$
while $(t_a > \overline{tp}_k)$ **do**
  $d \leftarrow d - v_{c,k} \cdot (\overline{tp}_k - t)$
  $t \leftarrow \overline{tp}_k$
  $t_a \leftarrow t + (d/v_{c,k+1})$
  $k \leftarrow k + 1$
end while
Return `travel_time` $= (t_a - t_d)$

3.2. *Mathematical formulation*

This section describes a Mixed Integer Programming (MIP) formulation for the TD-OP, based on the MIP for the OP [45]. The decision variables and parameters used in this model are listed below:
$x_{i,j,t}$ =1 if a vehicle traverses the arc (i, j) with a departure time in time slot t, 0 otherwise
$w_{i,j,t}$: departure time in time slot t when traveling from i to j
$\theta_{i,j,t}$: slope coefficient of the linear time-dependent travel time as defined in Equation 3a
$\eta_{i,j,t}$: intercept coefficient of the linear time-dependent travel time as defined in Equation 3b
$\tau_{i,j,t}$: lower limit of time slot t for arc (i, j)
$T_{i,j}$: number of time slots for arc (i, j) given by Algorithm 2
S_i: score of vertex i
t_{max}: maximum total travel time
vertex 1 is the start depot and vertex N is the end depot

$$\text{Max} \sum_{i=2}^{N-1} \sum_{j=2}^{N} \sum_{t=1}^{T_{i,j}} S_i x_{i,j,t} \tag{1a}$$

$$\sum_{j=2}^{N} x_{1,j,1} = \sum_{i=1}^{N-1} \sum_{t=1}^{T_{i,N}} x_{i,N,t} = 1 \tag{1b}$$

$$\sum_{i=1}^{N-1} \sum_{t=1}^{T_{i,h}} x_{i,h,t} = \sum_{j=2}^{N} \sum_{t=1}^{T_{h,j}} x_{h,j,t} \leq 1 \qquad \forall h = 2, ..., N-1 \tag{1c}$$

$$\sum_{i=1}^{N-1} \sum_{t=1}^{T_{i,h}} \left[w_{i,h,t} + (\theta_{i,h,t} \cdot w_{i,h,t} + \eta_{i,h,t} \cdot x_{i,h,t}) \right] = \sum_{j=2}^{N} \sum_{t=1}^{T_{h,j}} w_{h,j,t} \qquad \forall h = 2, ..., N-1 \tag{1d}$$

$$x_{i,j,t} \cdot \tau_{i,j,t} \leq w_{i,j,t} \leq x_{i,j,t} \cdot \tau_{i,j,t+1} \quad i = 1, ..., N-1, j = 2, ...N, \forall t \tag{1e}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} \sum_{t=1}^{T_{i,j}} \left[\theta_{i,j,t} \cdot w_{i,j,t} + \eta_{i,j,t} \cdot x_{i,j,t} \right] \leq t_{max} \tag{1f}$$

$$w_{1,i,1} = 0 \qquad \forall i = 1, ..., N \tag{1g}$$

$$x_{i,j,t} \in (0,1)\,;\, 0 \leq w_{i,j,t} \leq t_{max} \qquad \forall t, i, j = 1, ..., N \tag{1h}$$

The decision variable $x_{i,j,t}$ is equal to 1 when traveling from i to j with a departure time in time slot t, and equals 0 otherwise. $w_{i,j,t}$ is a continuous decision variable which contains the departure time at vertex i when traveling to vertex j in time slot t. Note that both decision variables, $x_{i,j,t}$ and $w_{i,j,t}$, are equal to zero when there is no path from i to j in timeslot t in the solution. In short, $w_{i,j,t}$ becomes a proxy for $x_{i,j,t}$ when checking if a path is scheduled between two vertices in timeslot t. This is a useful property to avoid the multiplication of decision variables in constraints (1d) and (1f).

The objective function and the first constraint are similar to the time-*independent* orienteering problem apart from the fact that they need to be applied for all time slots t. The objective function (1a) maximizes the total collected score. Constraint (1b) guarantees that the path starts in vertex 1 and ends in vertex N. Constraints (1c) and (1d) ensure the connectivity of both path and travel time. More specifically constraint (1c) guarantees that every vertex is visited at most once. Next, constraint (1d) guarantees that the departure time of a succeeding vertex in the route is equal to the sum of the departure time of the previous vertex together with the travel time between these two vertices. The travel time is calculated as a linear function. In this function the departure time is multiplied with a parameter $\theta_{i,j,t}$, whereafter parameter $\eta_{i,j,t}$ is added. The main difference with the regular MIP formulation of the OP lies in constraints (1e) and (1f) as the fixed distance is replaced by the time-dependent travel time. Constraint (1e) categorizes the departure time in the right time slot which is necessary to multiply the departure time with its corresponding θ and η in constraint (1f).

In addition to this, the following assumptions were made: a route must start in time slot one (constraint 1g) and no waiting is allowed (constraint 1d). These assumptions are motivated by the fact that our travel times are generated by a speed model that is conform the FIFO principle (Section 3.1) which implies that waiting has no benefit, nor in theory, nor in practice.

To execute this MIP, we need to determine an efficient set of time cut off points (*time slots*) for each arc ($\tau_{i,j,t}$). Please note the difference between *time slots* and *time periods* in the following paragraphs. A set of $K = 4$ different time periods is used to define different speed values per arc category and per day segment in Section 3.1. The exact number of time slots is unique per arc and depends on the number of *time periods* (K) and the actual values of the speed matrix. Each time an increase or decrease is found in the travel time between i to j, this moment in time will be stored. This is stored as the lower limit of the time slot called $\tau_{i,j,t}$, together with two corresponding linear regression coefficients ($\theta_{i,j,t}$ and $\eta_{i,j,t}$). These linear regression coefficients allow the travel time to be calculated as follows:

$$\texttt{travel\_time}_{i,j,w_{i,j,t}} = \theta_{i,j,t} * w_{i,j,t} + \eta_{i,j,t} \tag{2}$$

The procedure to find the efficient set of *time slots* consists of a combination of a forward arrival time calculation and a backwards departure time calculation when an upper *time period* is crossed. This procedure is displayed in the pseudocode of Algorithm 2.

At the start of this procedure, the variable `border` is equal to the lower limit of the first *time period* ($\underline{tp}_{k=1}$). Subsequently we calculate the arrival time (t_a) if we depart at `border`. The `border` is stored as a *time slot* limit together with the corresponding travel time. Subsequently `border` is increased to the lower limit of the next *time period* $k+1$.

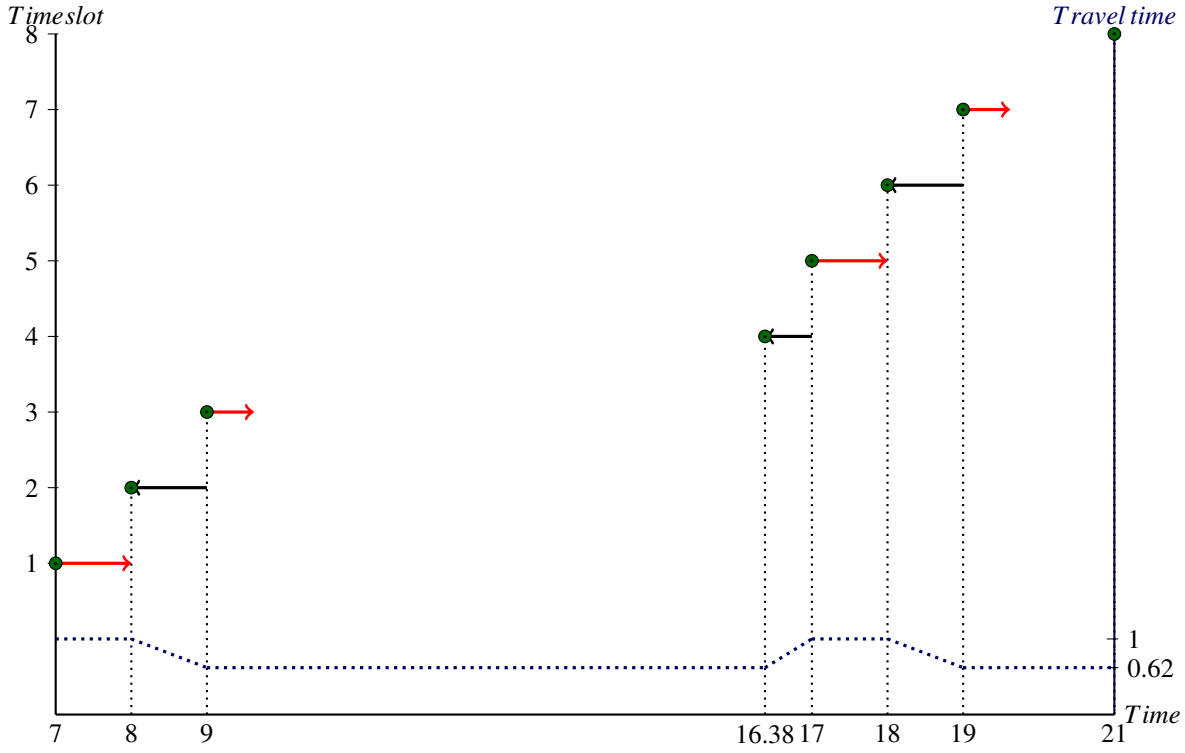
Next, a while loop is executed until `border` is equal to $\underline{tp}_K$ (lower limit of the last time period). First, we calculate the departure time (t_d) that enables t_a to be equal to `border`. The t_d and travel time are again stored as lower limit with corresponding travel time. Subsequently we calculate the t_a when we depart at `border` and store this information. Finally, `border` is set equal to the lower limit of the next time period $k+1$.

This procedure is repeated for every arc and is visually represented for one arc in Figure 1 using the speed matrix from Section 3.1. On this figure the x-axis represents the time divided into K time periods. On the left y-axis the time slots and on the right y-axis the travel time are displayed. The green dots represent the time slot limits that are stored in the travel time matrix. The first green dot lying on the red line is calculated before the start of the while loop. The green dots lying on the black lines correspond to the backward departure time calculations at the start of the while loop. The green dots on the other red lines correspond to the succeeding forward arrival time calculations. As you can see the green dots correspond with the start of an increase/decrease in the travel time function.

Algorithm 2 Time slot finder algorithm

for all pairs (i,j) **do**
  $t = 1, k = 1$
  `border` $= \underline{tp}_k$
  Calculate t_a when $t_d =$ `border`
  Store `border` as $\tau_{i,j,t}$ together with the $\texttt{travel\_time}_t = t_a - t_d$
  $t = t + 1$
  $k = k + 1$
  `border` $= \underline{tp}_k$
  while `border` $< \underline{tp}_K$ **do**
    Calculate the corresponding t_d when $t_a =$ `border`: $t_d =$ `border` $- \texttt{travel\_time}_{t-1}$
    Store t_d as $\tau_{i,j,t}$ together with the $\texttt{travel\_time}_t$
    $t = t + 1$
    Calculate the corresponding t_a when $t_d =$ `border`
    Store `border` as $\tau_{i,j,t}$ together with the $\texttt{travel\_time}_t$
    $t = t + 1$
    $k = k + 1$
    `border` $= \underline{tp}_k$
  end while
  Store `border` as $\tau_{i,j,t}$ together with the $\texttt{travel\_time}_{t-1}$
  $T_{i,j} = t - 1$
end for

Figure 1: An example of finding an efficient set of time slots for an arc with $d = 0.5$ and belonging to arc category 1



Based on the travel time information and the efficient set of time slot limits ($\tau_{i,j,t}$), $\theta_{i,j,t}$ and $\eta_{i,j,t}$ can be calculated as follows:

$$\theta_{i,j,t} = \frac{\texttt{travel\_time}_{t+1} - \texttt{travel\_time}_{t}}{\tau_{i,j,t+1} - \tau_{i,j,t}} \tag{3a}$$

$$\eta_{i,j,t} = \texttt{travel\_time}_{t} - \theta_{i,j,t} * \tau_{i,j,t} \tag{3b}$$

3.3. TD-OP Datasets

To test the developed solution procedures, adequate datasets and appropriate performance measures are needed. So far no benchmark datasets have been developed for the time-dependent orienteering problem. Therefore, we have created new data sets based on the ones available for the (time-independent) team orienteering problem [6, 45].

To create a time-dependent data set every arc needs to be assigned to an arc category. The datasets commonly used to test TD-VRP solution methods have been generated by randomly assigning arcs to an arc category [19, 9, 18, 43].

This can lead to the situation depicted in Figure 2 and to datasets that might not be challenging enough to test a solution method that is designed to execute on congested networks. As can be seen in Figure 2, traveling from C to B or B to C results in the same evening peak category, furthermore there is no logical relationship between the morning and evening peaks. Thirdly, the always busy category from B to A is easily bypassed through C in the morning.

Evaluating solution procedures based on datasets where arcs are randomly assigned to arc categories might therefore lead to incorrect performance conclusions as it is, for instance, easier than in practice to find alternative routings, which requires less sophisticated algorithms. This issue is also mentioned by Figliozzi [10] who developed a set of accessible datasets for the TD-VRP. However, in these datasets of Figliozzi [10], the same speed profile is used for all arcs, which also might lead to less challenging test problems and therefore a different procedure is developed in this paper. To avoid a random assignment of arcs to arc categories, existing datasets of the team orienteering problem have been transformed to time-dependent instances by assigning the arcs in an intelligent way to a congestion pattern. Although most of the work could be automated some manual assignments are still needed to make the resulting datasets much more realistic, as will be explained below.

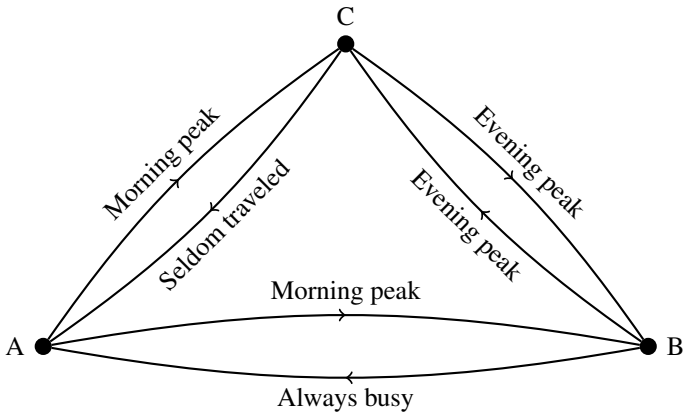


Figure 2: Arcs randomly assigned to an arc category

More specifically, seven well-known graphs, first published by Chao et al. [6] for the team orienteering problem, have been transformed to seven TD-OP datasets. First, city centers, highways and residential zones are arbitrarily marked on each of the seven graphs (one graph per instance). Thereafter, arcs were assigned to one of the five categories based on a number of rules concerning these zones. The four basic rules that were used are:

- Arcs situated completely in a city center are assigned to the *always busy* category;
- Arcs situated completely in the highway zones are added to the *two peaks* category;
- Traveling from a residential area to a city center results in a *morning peak* congestion and an *evening peak* is observed when traveling in the opposite direction;
- Arcs that run from or to a vertex which is not a member of a certain zone are assigned to the *seldom traveled* category.

Examples of these rules, and other types of assignments, are displayed in Figure 3. This figure also demonstrates that, in practice, a specific combination of two zones not always results in the same arc category between these zones. Especially arcs between vertices situated in non-connected zones (non-neighboring zones) have to be treated differently than arcs of vertices belonging to connected zones. For example, traveling from a vertex in a highway zone to a vertex in a nearby commuting zone results in the evening peak category, however traveling from that same vertex to a remote commuting zone member results in the seldom traveled category. As a result, we decided to assign a limited number of arcs based on the author's insight, in order to enhance the realistic representation of the datasets.

Subsequently it is assumed that one-day trips are planned, starting at the start depot at 7 am and ending at the end depot before 9 pm, allowing for a maximum travel time of 14 hours. Varying this available travel time (t_{max}) within a certain range, allows to create multiple instances based on the same graph information, i.e, the same vertices, arcs and time periods. In order to ensure that the instances are difficult enough and that enough vertices can be visited in a time-dependent instance with a lower maximum travel time, the relative distance between the vertices has been rescaled by a specific factor per dataset.

The importance of this rescaling is supported by Vansteenwegen [44], who indicates that the most difficult (time-independent) OP instances are those for which the selected number of vertices is slightly more than half of the total number. If the time budget allows the selection of half of the vertices, the largest possible number of selections will have to be evaluated by the algorithm. Moreover, determining a path between the selected vertices becomes more time consuming when the number of vertices increases. Therefore, creating instances which contain this property is important to ensure that the benchmark instances are challenging. All datasets, together with the developed solution procedure, are available at: http://www.mech.kuleuven.be/en/cib/op/.

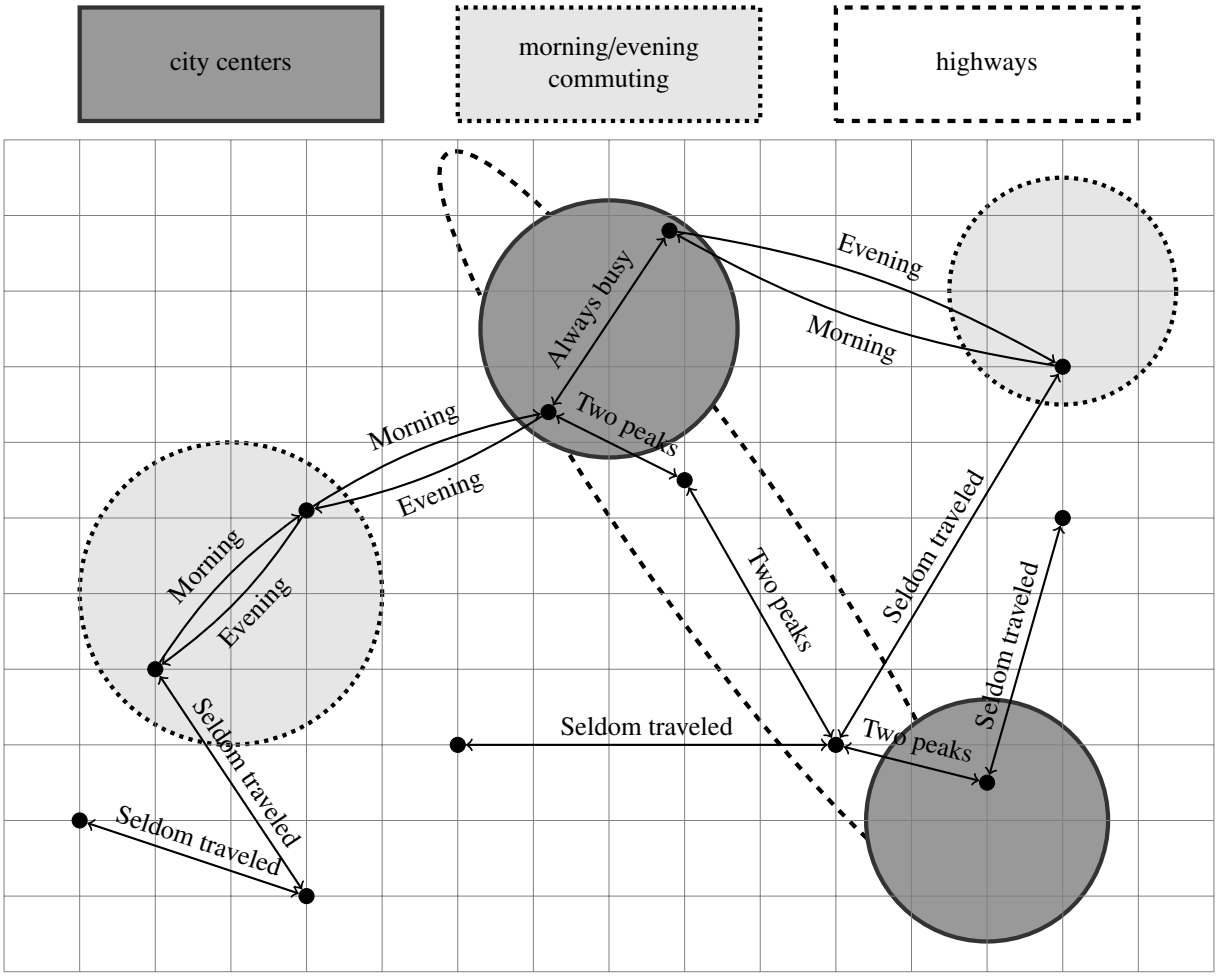


Figure 3: Fictive example of a time-dependent orienteering problem instance

Note that the instance development procedure implies that arcs are often asymmetrically added to an arc category (e.g. morning peak from A to B and evening peak from B to A). This has considerable implications for the travel time calculation and the local search procedures (Section 4.1 & 4.2).

4. Solution approach

Since high-quality solutions are required and the computation time should be limited to only a few seconds, the literature on vehicle routing suggests the implementation of a local search based metaheuristic [39]. In this paper, a metaheuristic, based on the principles of an ant colony system (ACS), is implemented to tackle the TD-OP. This choice was motivated by the fact that generally very complex problems require simple solution frameworks and Bullnheimer et al. [4] state that the ACS produces starting vehicle routing problem solutions that are more easily improved by a local search procedure than starting solutions produced by a genetic algorithm. The ACS is based on the ant colony optimization algorithm (ACO) of Ke et al. [23] and Schilde et al. [37] for the time-*independent* OP and on the ACS of Donati et al. [9] for the TD-VRP.

In order to solve the TD-OP, as for every metaheuristic, a good balance between intensification and diversification is essential [40]. Therefore, a specific *insert* procedure is designed in order to intensify the search for improvement. The strength of this insertion step lies in the fast evaluation of the possible insertion of a vertex. In order to diversify the search, each metaheuristic framework has a specific manner to escape from local optima. The pheromone trails

in the ACS are depreciated ("evaporated") during the construction procedure. The ACS diversification mechanism is discussed in more detail in Section 4.3.

The remainder of this section is organized as follows. First, the local search procedures are discussed: the insert local search procedure is explained in Section 4.1 and the modified 2-Opt procedure in Section 4.2. The ACS framework is explained in Section 4.3.

4.1. Insert local search procedure

The problem-specific *insert* local search procedure iteratively attempts to insert non-included vertices into an existing solution, thus improving its total score. To prevent a full and time-consuming evaluation of a solution after every insertion attempt, we store for every vertex in the current solution the maximum amount of time that a visit to it can be postponed before the solution becomes infeasible (`max_shift`). This enables an efficient checking and updating mechanism. The `max_shift` metric can be calculated from the last vertex to the first vertex in the solution and only needs to be updated for some vertices when an extra vertex is actually included in the solution, not when it is only considered for inclusion during the insert procedure. A similar method was discussed in [9] and the calculation is presented in pseudo code in Algorithm 3. Finally the procedure is also visually presented in Figure 4.

Algorithm 3 Calculation of `max_shift` - input: s

$t_a \leftarrow \underline{tp}_1 + t_{max}$
for every vertex i in solution s except the start depot & end depot **do**
  $z \leftarrow s_{i+1}$
  $y \leftarrow s_i$
  find k such that $t_a > \underline{tp}_k$
  $t_d \leftarrow t_a$
  `remaining_distance` $\leftarrow d_{y,z}$
  `distance_covered` $\leftarrow (t_a - \underline{tp}_k) * v_{y,z,k}$
  while (`remaining_distance` > `distance_covered`) and ($k > 1$) **do**
    `remaining_distance` $\leftarrow$ `remaining_distance` $-$ `distance_covered`
    $t_d \leftarrow t_k$
    `distance_covered` $\leftarrow (\underline{tp}_k - \underline{tp}_{k-1}) \cdot v_{y,z,k-1}$
    $k \leftarrow k - 1$
  end while
  $t_d \leftarrow t_d -$ `remaining_distance`$/v_{y,z,k}$
  `max_shift`$_i \leftarrow (t_d -$ `actual_departure_time`$_i)$
  $t_a \leftarrow t_d$
end for

When `max_shift` has been calculated for every vertex in the current solution, a list called `include` is created of all vertices that are not yet included in the solution. The vertices are added to this list in random order.

The insert procedure tries to insert a vertex from this list into the current solution, if the extra travel time required to visit this new vertex is smaller than the value of `max_shift` of the succeeding vertex. For example, when the algorithm attempts to insert vertex y (member of the `include` list) between x and z, the extra time-dependent travel time equals:

$$\Delta\texttt{travel\_time} = \texttt{travel\_time}_{x,y} + \texttt{travel\_time}_{y,z} - \texttt{travel\_time}_{x,z} \tag{4}$$

Vertex y can only be inserted into the current solution when the extra travel time is smaller than or equal to the maximum amount of time vertex z can be shifted to a later moment in time or:

$$\Delta\texttt{travel\_time} \leq \texttt{max\_shift}_z \tag{5}$$

When vertex y is actually included in the solution, the algorithm updates the travel time from x to y, as well as the travel times between the vertices succeeding vertex y. This is necessary, because the insertion of vertex y has most likely caused a change in travel time for the arcs of the solution succeeding vertex y.

The max_shift of the vertices succeeding vertex y can easily be adjusted (marked in green) based on the previous max_shift value, the previous travel time and the new travel time of the vertex under consideration based on the following equation :

$$\texttt{max\_shift}_i = \texttt{previous\_max\_shift}_i + (\texttt{previous\_travel\_time}_i - \texttt{new\_travel\_time}_i) \tag{6}$$

In this equation $\texttt{travel\_time}_i$ represents the travel time to reach i from its immediate predecessor. Second, a recalculation of max_shift for the vertices preceding vertex z is also necessary (marked in red).

As a result of this procedure, the computation time needed to check if a vertex can be included or not, is drastically reduced. Nevertheless, after an actual insertion, a complete recalculation of the new solution is required.

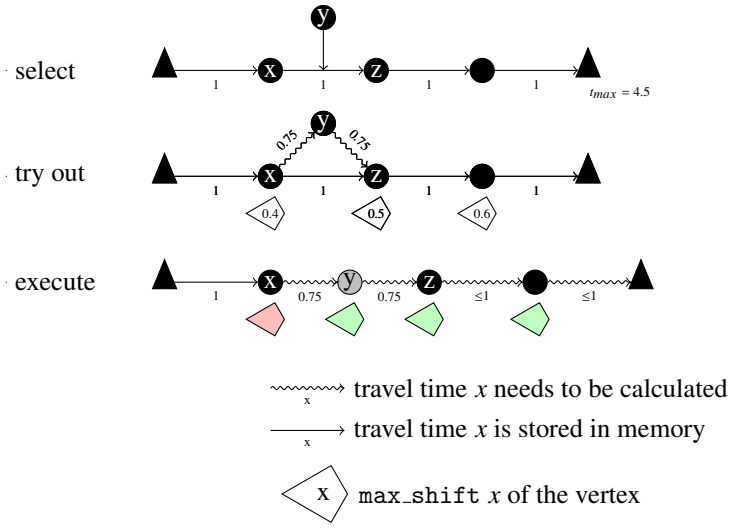


Figure 4: The time-dependent insert move

Another issue that concerns the insert local search move is the triangle inequality property, which for road networks means that traveling from vertex A to C is shorter than traveling from A to B and from B to C. In a time-dependent problem however, due to traffic jams on the direct road between A and C, it might be actually faster to drive through B. The result of wrongly assuming that this property is still valid in a time-dependent setting is that one may not consider potential changes of the shortest paths due to varying travel times as mentioned by Donati et al. [9] and Fleischmann et al. [12]. In practice the triangle property will always be respected. If traveling from A to B would be faster through C, the travel time of the path through C would be used as the travel time between A and B. To model this in a proper way, that would mean that we have to check all possible triangles [A,B,C] for each possible departure time at A. We decided not to focus on this issue and we refer to [25] for a possible implementation.

Therefore, inserting a vertex might shorten the travel time of the solution. If for example the sequence AB, displayed in Figure 5 is included in the solution the insert move might insert vertex C in between A en B which results in a decrease in travel time of 3.51 time units:

$$\texttt{travel\_time}_{A,B,t_d} > \texttt{travel\_time}_{A,C,t_d} + \texttt{travel\_time}_{C,B,[t_d+\texttt{travel\_time}_{A,C,t_d}]}$$

$$\texttt{travel\_time}_{A,B,7} > \texttt{travel\_time}_{A,C,7} + \texttt{travel\_time}_{C,B,[7+\texttt{travel\_time}_{A,C,t_d}]}$$

$$(1/0.5) + (5/0.81) > (3/1.5) + \texttt{travel\_time}_{C,B,9}$$

$$8.17 > 2 + (4/1.5)$$

$$8.17 > 4.66$$

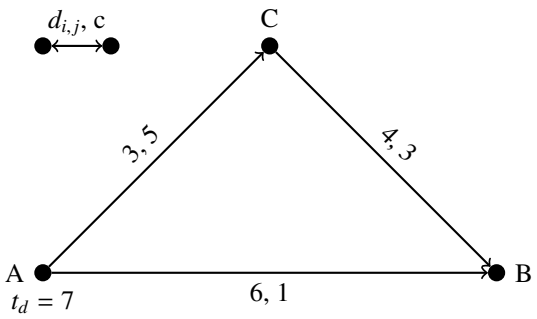


Figure 5: An example where the triangle property is not valid for the time- dependent travel times

4.2. Modified 2-Opt

The basic 2-Opt procedure removes two arcs and tries to replace them with two new arcs not previously included in the path. If this procedure reduces the travel time of the solution, the new solution is accepted, otherwise the original solution is retained [45]. This procedure cannot be easily used in a time-dependent environment because the direction of the part of the solution between the removed arcs is reversed. Traversing arcs in the opposite direction demands a complete and time-expensive recalculation of the solution. It might be possible that arcs that are troubled by morning congestion and therefore originally scheduled in the evening, are reinserted into the morning by a 2-Opt procedure and therefore make the whole solution infeasible or at least undesirable.

To circumvent these time-expensive calculations, the algorithm first preselects interesting arcs based on the gain in distance. The most interesting arc couple is then evaluated based on real travel times. If the new solution is feasible, it is accepted. It should be noted that it is not required that the travel time of the new solution is smaller than the travel time of the previous solution. In the algorithm developed in this paper, this 2-Opt procedure is merely used as a diversification procedure, rather than to increase the quality of a solution.

4.3. Ant colony system

The ACS is based on the behavior of a foraging ant colony. It is a constructive metaheuristic that constructs several solutions independently (each construction procedure is represented by an agent commonly called an "ant") and uses memory structures called "pheromones trails" to mark traveled arcs and communicate between the different ants. The ACS framework is displayed in Algorithm 4, together with the corresponding input parameters and variables. s_{ib} is the best solution of the current `iteration`, s_{gb} represents the best solution found during the entire optimization procedure and $F(s_x)$ refers to the objective function (total score) of solution s_x.

Algorithm 4 Ant colony system - input parameters: $\alpha, \beta^*, \rho, \texttt{max\_ants}, \tau_{init}, N_{ni}^{max}$

$s_{ib} \leftarrow 0$, $s_{gb} \leftarrow 0$, $N_{ni} \leftarrow 0$, $\texttt{iteration} \leftarrow 0$
while $\texttt{iteration} < N_c$ **do**
  Initialize τ, η (τ_{init})
  Construct initial solutions:
  for $i \leftarrow 1$ to $\texttt{max\_ants}$ **do**
    Construct solution (τ, η, α, β)
    Local pheromone update (τ, ρ, s_{ib})
    2-Opt
    Insert ($\texttt{max\_shift}$)
  end for
  $s_{ib} \leftarrow \arg\max(F(s_1), F(s_2), ..., F(s_{\texttt{max\_ants}}))$
  if $F(s_{ib}) > F(s_{gb})$ **then**
    $s_{gb} \leftarrow s_{ib}$
    $N_{ni} \leftarrow 0$
  else
    $N_{ni} \leftarrow N_{ni} + 1$
  end if
  Global pheromone update (τ, s_{ib}, N_{ni}, N_{ni}^{max})
  $\texttt{iteration} \leftarrow \texttt{iteration} + 1$
end while

Before the start of the ACS, the value of the greedy information, $\eta_{i,j}$ for arc (i, j) is calculated as the ratio of the score of vertex j and the travel distance to reach vertex j from vertex i. This way of working turns out to be less computational expensive in comparison to using the ratio of the score and the time-dependent travel time to the next vertex. The pheromone value ($\tau_{i,j}$) of all arcs is initially set at a value τ_{init}.

The construct solution procedure creates $\texttt{max\_ants}$ solutions independently and sequentially. Each construction starts from an empty solution and adds vertices at the end of the solution until no more vertices can be inserted due to the travel time restriction. At that point, the end vertex is added to finalize the solution, and the algorithm moves on to the next solution until $\texttt{max\_ants}$ solutions are created. Before adding a vertex following the already included vertex u, a list called C_u of all vertices feasible to include is created. This is done by calculating the time-dependent travel time between the last included vertex u and a vertex under consideration plus the travel time between the vertex under consideration and the end vertex. If the sum of both travel times together with the total travel time of the solution is smaller than t_{max}, the vertex under consideration can be added to the solution. Afterwards, each vertex in the list receives a probability to be included. This probability (l_i) that vertex i will be added to the solution is calculated as follows:

$$l_i = \frac{\tau_{u,i}^{\alpha} \cdot \eta_{u,i}^{\beta}}{\sum_{w \in C_u} \tau_{u,w}^{\alpha} \cdot \eta_{u,w}^{\beta}} \qquad \forall i \in C_u \tag{7}$$

In this equation, α determines how much weight is given to the pheromone value and β defines how much weight is given to the greedy information. Then, a random number is generated to determine, together with the probability l_i, which vertex from the list is added to the solution by using a roulette wheel selection method. Note that the relative values of α and β together determine how much weight is given to the randomness in the selection procedure. For example, the random numbers will have less impact if α and β both equal 3 than when they both equal 1, as the difference in τ or η values of feasible vertices is disproportionately reflected in their l_i values. A vertex that has a slightly better value of τ than another vertex will have a disproportional larger l_i value and thus a higher probability to be selected. In other words, the procedure becomes more greedy. Subsequently, the τ value related to the newly added arc in the solution is decreased. This local pheromone procedure which enhances diversification is executed as follows:

$$\tau_{u,i} = \tau_{u,i} \cdot (1 - \rho) \tag{8}$$

ρ is called the evaporation rate and is usual set at a rather low value (e.g. 0.05). The evaporation procedure attempts to enhance the diversity of the solution procedure by preventing the same arcs from being added to a large number of solutions. Note that the pheromone values should be prevented from becoming very small, leading to a possible (near) division by zero in Equation 7. When a τ-value is close to zero, it is reset to τ_{init}. After the construction of `max_ants` solutions, the insert and 2-Opt procedures are executed on all constructed solutions. The modified 2-Opt procedure evaluates all possible 2-Opt moves but only the most interesting move is executed. Since executing a 2-Opt move is computationally very expensive, the procedure is stopped after one execution in order to limit the computation time. The insert procedure however, is executed in a first-improving manner and stops when no more feasible improvements can be found.

The solution of each iteration with the highest score is stored (s_{ib}) and arcs that are used in this solution are made more attractive to be used in the solution construction procedure of succeeding iterations, by increasing their corresponding pheromone value:

$$\tau_{i-1,i} \leftarrow \tau_{i-1,i} + \tau_{init} \qquad \forall i \in s_{ib} \,|\, i > 1 \tag{9}$$

This makes it more likely that these arcs will be used in a subsequent construction procedure (intensification). If the score (s_{ib}) is better than the global best score found during previous iterations (s_{gb}), s_{gb} is updated. Finally, these steps are repeated N_c times but to prevent that only a couple of arcs dominate in the solution construction procedure (local optima), the pheromone values are reset to τ_{init} when no improvement can be found during a certain number of iterations (N_{ni}^{max}). This means that all arcs have again an equal probability to be chosen during the next construction procedure, allowing diversification.

5. Computational experiments

5.1. Comparison to optimal solutions of small instances

The first experiment consists out of a straightforward comparison of the results of the ACS and the optimal solution found by solving the MIP from Section 3.2. However, most of the time-dependent instances of Section 3.3 are too complex to be solved with a commercial solver using the MIP formulation developed in this paper. Since this is not the focus of our research only a basic implementation of this MIP was implemented using the commercial solver, CPLEX 12.5 (64-bit) on a computer with an i5 2.6 GHz processor and 8 GB of memory. The choice to apply this commercial solver to the MILP of the TD-OP was motivated by the fact that commercial solvers like CPLEX and Gurobi are known to be very effective in tackling this kind of problems. We refer to the exact solution methods of [11, 2, 20] for variants of the OP and CPLEX is also used for vehicle routing problems in a time dependent context by Kok et al. [24]. The restriction of 48 hours (!) of computation time per instance allowed to find an optimal solution for most of the instances of the first 3 datasets. The comparison between the results of the ACS and these optimal solutions for five independent runs of the algorithm is displayed in Table 2. In the name of each instance, the number refers to the graph, originally developed by Chao et al. [6] that is used. The characters a to i refer to increasing values of t_{max}.

As a performance metric, the percentage gap between the total score of the optimal solution and the total score of the heuristic solution is used, together with the CPU time:

$$\text{gap} = \frac{\text{optimal score} - \text{heuristic score}}{\text{optimal score}} \tag{10}$$

Both the average gap and the minimum and maximum gap on all instances are recorded during our analysis.

Table 2: Comparison to optimal solutions of small instances

			CPLEX		ACS			
	N	t_{max} hour	optimal score	cpu s	best %gap	avg %gap	worst %gap	cpu sec
1.a	32	5	115	41	**0.0**	**0.0**	**0.0**	0.2
1.b	32	6	135	143	**0.0**	**0.0**	**0.0**	0.1
1.c	32	7	160	351	**0.0**	**0.0**	**0.0**	0.1
1.d	32	8	185	515	**0.0**	2.2	5.4	0.1
1.e	32	9	210	687	**0.0**	0.5	2.4	0.1
1.f	32	10	230	37514	**0.0**	0.4	2.2	0.1
1.g	32	11	250	6787	**0.0**	**0.0**	**0.0**	0.1
1.h	32	12	270	70225	1.9	1.9	1.9	0.1
2.a	21	5	100	11	**0.0**	**0.0**	**0.0**	0.0
2.b	21	6	150	24	**0.0**	**0.0**	**0.0**	0.0
2.c	21	7	195	35	**0.0**	**0.0**	**0.0**	0.1
2.d	21	8	220	49	**0.0**	**0.0**	**0.0**	0.1
2.e	21	9	260	126	**0.0**	**0.0**	**0.0**	0.1
2.f	21	10	310	383	**0.0**	**0.0**	**0.0**	0.1
2.g	21	11	340	671	**0.0**	**0.0**	**0.0**	0.1
2.h	21	12	375	5356	**0.0**	**0.0**	**0.0**	0.1
2.i	21	13	425	14217	**0.0**	**0.0**	**0.0**	0.1
3.a	33	5.5	370	96	**0.0**	4.3	5.4	0.1
3.b	33	6.5	420	272	**0.0**	**0.0**	**0.0**	0.1
3.c	33	7.5	500	371	4.0	4.0	4.0	0.1
3.d	33	8.5	560	503	**0.0**	2.9	3.6	0.1
3.e	33	9.5	620	662	**0.0**	1.9	4.8	0.2
3.f	33	10.5	650	1928	**0.0**	0.3	1.5	0.1
3.g	33	11.5	690	16704	**0.0**	1.2	2.9	0.2
				max	4.0	4.3	5.4	0.2
				avg	0.2	0.7	1.3	0.1
				% optimal	92.6	63.0	63.0	

The results prove the validity of our MIP model and the excessive time needed to solve these small instances using an exact solution approach. The required computation time depends heavily on the number of time slots needed per arc and they exponentially increase with the value of t_{max}. Furthermore, this table illustrates the near-optimal results and the very short computation times of the ACS on small problem instances.

5.2. Comparison to known optimal solutions of large instances

To evaluate the performance of the metaheuristic developed in this paper, it would be useful to also have optimal solutions for larger benchmark instances. Therefore all instances were solved first as time-*independent* OPs using the MIP provided by Vansteenwegen et al. [45]. During this optimization, the travel time was calculated using, on each arc, the maximum speed of its corresponding arc category. Still not all time-*independent* OP instances could be solved due to time and memory limitations. The excessive CPU time needed to solve these time-*independent* OP instances (not displayed), often more than 100 hours, illustrates again that it would be pointless to solve large time-dependent OPs to optimality. The found optimal scores are displayed in Table 3 (column "optimal").

Following the optimization by CPLEX, the optimal time-*independent* solutions (sequence of vertices) are used to modify the original time-dependent instances in such a way that slightly modified time-dependent instances with known optimal solution are created.

More specifically, for each arc included in the optimal time-independent solution, the travel speed is set to its maximal value, but only during the time periods these arcs are traversed. Since the travel speed is only modified in some of the time periods of these arcs, these arcs still have time-dependent travel times. The time-dependent travel times on all other arcs are not modified. In this way, it is ensured artificially that the time-independent optimal solution is also an optimal solution to the modified time-dependent instance.

The procedure is explained in detail with a simple problem instance in Figure 6. In part A the optimal time-independent solution found by CPLEX using the MIP model from Vansteenwegen et al. [45] is displayed. In the next part, the travel speed is set to its maximal value (depending on the arc category) for the selected arcs in part A and only during the time periods they are used in the time-independent solution. Second, the time-dependent arcs have different characteristics for each traveling direction. In part C the ACS is executed on this adapted problem instance. Furthermore, note that the found solution in part C has a lower objective value than the optimal solution in part B which results in a gap of 16.7%.

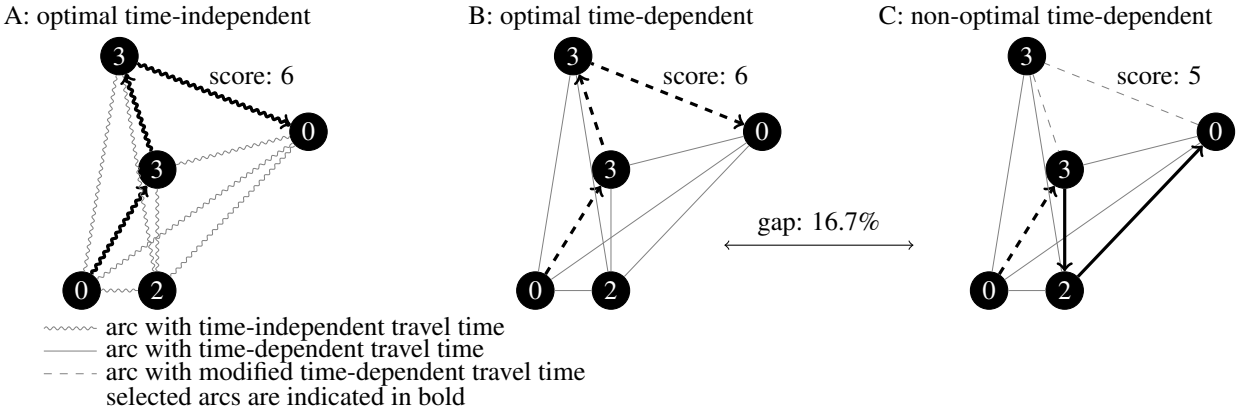


Figure 6: Example of creating a time-dependent instance with a known optimal solution

The creation of the benchmark instances in this way allows a comparison of the performance of the developed solution methods with known optimal solutions for larger instances. For five independent runs of the algorithm, these results are displayed in Table 3. In Table 4 the % gap per dataset is displayed. The effect of the parameters that are used in the metaheuristic and their exact values used in these experiments are discussed in detail in Section 5.3.

Table 3: Comparison to known optimal solutions of large instances

	N	t_{max} hours	optimal score	best % gap	avg % gap	worst % gap	CPU s
1.a	32	5	135	**0.0**	**0.0**	**0.0**	0.1
1.b	32	6	165	**0.0**	**0.0**	**0.0**	0.1
1.c	32	7	185	**0.0**	**0.0**	**0.0**	0.1
1.d	32	8	210	**0.0**	**0.0**	**0.0**	0.1
1.e	32	9	240	**0.0**	**0.0**	**0.0**	0.1
1.f	32	10	260	**0.0**	1.5	1.9	0.1
1.g	32	11	275	**0.0**	**0.0**	**0.0**	0.1
1.h	32	12	285	**0.0**	**0.0**	**0.0**	0.1
1.i	32	13	285	**0.0**	**0.0**	**0.0**	0.1
2.a	21	5	165	**0.0**	**0.0**	**0.0**	0.0
2.b	21	6	200	**0.0**	**0.0**	**0.0**	0.1
2.c	21	7	225	**0.0**	**0.0**	**0.0**	0.1
2.d	21	8	275	**0.0**	**0.0**	**0.0**	0.1
2.e	21	9	315	**0.0**	**0.0**	**0.0**	0.1

2.f	21	10	375	**0.0**	**0.0**	**0.0**	0.1
2.g	21	11	415	**0.0**	**0.0**	**0.0**	0.1
2.h	21	12	440	**0.0**	**0.0**	**0.0**	0.1
2.i	21	13	450	**0.0**	**0.0**	**0.0**	0.1
3.a	33	5.5	430	**0.0**	**0.0**	**0.0**	0.1
3.b	33	6.5	490	**0.0**	**0.0**	**0.0**	0.1
3.c	33	7.5	550	**0.0**	**0.0**	**0.0**	0.1
3.d	33	8.5	590	**0.0**	**0.0**	**0.0**	0.1
3.e	33	9.5	630	**0.0**	**0.0**	**0.0**	0.1
3.f	33	10.5	680	**0.0**	**0.0**	**0.0**	0.2
3.g	33	11.5	730	**0.0**	**0.0**	**0.0**	0.2
3.h	33	12.5	770	**0.0**	1.0	2.6	0.1
3.i	33	13.5	800	**0.0**	0.8	3.8	0.1
4.a	100	5	486	**0.0**	0.8	1.0	0.7
4.b	100	6	590	0.5	1.7	3.1	0.8
4.c	100	7	679	1.6	2.5	3.5	0.9
4.d	100	8	771	3.6	4.5	5.1	1.0
4.e	100	9	853	3.4	4.2	5.5	1.1
4.f	100	10	932	2.1	3.0	4.3	1.2
4.g	100	11	1007	**0.0**	0.9	4.1	1.2
4.h	100	12	1083	1.7	3.0	3.6	1.3
4.i	100	13	1147	1.3	1.7	2.4	1.3
4.j	100	14	1198	4.2	5.3	6.4	1.4
5.a	66	5.5	580	**0.0**	0.3	1.7	0.3
5.b	66	6	650	2.3	2.3	2.3	0.3
5.c	66	7	770	1.3	1.6	2.6	0.4
6.a	64	6.5	870	**0.0**	0.3	0.7	0.5
6.b	64	7	930	**0.0**	0.6	1.3	0.4
6.c	64	8	1056	**0.0**	0.9	2.3	0.5
6.d	64	9	1152	0.5	0.8	2.1	0.5
6.e	64	10	1236	**0.0**	1.1	1.9	0.6
6.f	64	11	1308	0.9	1.8	2.8	0.5
6.g	64	12	1344	0.9	1.5	2.2	0.7
6.h	64	13	1344	**0.0**	**0.0**	**0.0**	0.5
6.i	64	14	1344	**0.0**	**0.0**	**0.0**	0.5
7.a	102	4	532	4.5	5.3	7.0	0.7
7.b	102	5	648	4.9	6.1	9.0	0.8
7.c	102	6	774	0.8	1.1	1.7	1.0
7.d	102	7	884	1.7	1.8	2.1	1.1
7.e	102	8	994	4.7	6.1	7.0	1.1
7.f	102	9	1090	1.8	2.5	3.7	1.2
7.g	102	10	1175	1.2	3.1	4.9	1.3
7.h	102	11	1251	3.6	4.6	5.1	1.3
7.i	102	12	1317	2.7	3.2	3.5	1.4
7.j	102	13	1368	5.0	5.6	6.3	1.4
			max	5.0	6.1	9.0	1.4
			avg	1.0	1.4	2.0	0.5
			% optimal	61.0	44.1	44.1	

Table 4: % gap (average, best, worst) and average CPU time per dataset

Dataset	N	best % gap	avg % gap	worst % gap	CPU s
1	31	0.0	0.2	0.2	0.1
2	21	0.0	0.0	0.0	0.1
3	33	0.0	0.2	0.7	0.1
4	100	1.8	2.8	3.9	1.1
5	66	1.2	1.4	2.2	0.3
6	64	0.3	0.9	1.7	0.5
7	102	3.1	3.9	5.0	1.1

These results prove the high performance quality of the ACS, since the average gap is very low at 1.4%. Furthermore, the known optimal solution could be found for 26 out of 59 test instances.

A second conclusion is that the average gap increases as the test instances become more complex due to a longer travel time limit and an increasing number of vertices. Studying the computation time leads to the conclusion that the ACS is very fast, as on average only one second is needed to obtain a solution. The maximum observed CPU time was 1.6 seconds, which is more than fast enough for most application purposes. Therefore, it can be concluded that the ACS is able to deliver a high performance requiring a minimal computational effort.

To stress the validity of this second test procedure, the average gap on the same instances from Section 5.1 is 0.1% and the maximum gap equals 1.5% which does not deviate too much from the values of respectively 0.7% and 4.3% obtained in Section 5.1.

5.3. Sensitivity analysis

In this section, the impact of the input parameters and the design of the ACS is discussed. It is interesting to know how much the performance of the algorithm depends on the specific values of the parameters. A first input parameter that needs separate attention is the number of iterations (N_c) that each construction and improvement cycle is performed. Given the goal of obtaining solutions within a few seconds, the value of this parameter has been severely restricted, and only 10,000 trial solutions were allowed. The exact value of N_c is therefore adjusted to the value of `max_ants`. For example when 20 ants are used, 500 iterations were allowed in order to generate 10,000 trial solutions. Varying the value of N_c gives an indication on the likelihood of the metaheuristic to get stuck in local optima (lack of diversity). If the algorithm easily gets stuck at a local optimum, increasing the number of trial solutions would not lead to significantly better solutions.

The average results of five runs can be found in Table 5. From this table, it can be concluded that the results keep improving when the number of iterations is increased. Thus, the algorithm has an appropriate diversification strategy.

Table 5: Effect of the number of trial solutions on the average gap and average CPU time (s)

Amount of trial solutions	5,000	10,000	20,000	40,000	80,000
avg gap	2.0%	1.6%	1.2%	1.0%	0.8%
max gap	8.8%	8.7%	6.4%	6.0%	5.6%
avg time	0.3	0.5	1.2	2.0	4.4

Table 6 provides an overview of the used input parameters of ACS, including their value used for the experiments in Section 5.1 and 5.2. These values are based on preliminary experiments on a test set of 21 instances, constructed by randomly selecting 3 instances per dataset.

Table 6: Overview of the input parameters

Input parameter	Description	Value
τ_{init}	start pheromone value	1
α	importance of pheromone information	4
β^*	importance of heuristic information ($\beta = \beta^* \cdot N$)	0.07
`max_ants`	number of solutions that are constructed and improved per iteration	75
ρ	evaporation rate	0.01
N_{ni}^{max}	maximum number of iterations without improvement (% of N_c)	25%

A second effect which needs to be analyzed is the robustness of the algorithm when its parameters are changed. Therefore, the orthogonal combinations displayed in Table 7 for the ACS were tested on the same random sample set of 21 test instances after 10,000 trial solutions. To eliminate the effect of the random numbers generator, each configuration was executed 5 times and the average gap was used to benchmark the performance. Tables 8 shows the impact of deviating from the currently used value. The `total` %gap corresponds to the average gap over all orthogonal combinations.

Note that preliminary tests indicated that the value of τ_{init} has no impact on the results. Furthermore, these tests showed that the optimal β parameter strongly depends on the size of the problem instance therefore the altered values of β represent a percentage, β^*, of N.

Table 7: Tested orthogonal combinations

Input parameter	Values
α	1, 3, 6, 9
β^*	0.01, 0.03, 0.05, 0.1
`max_ants`	10, 25, 50, 75, 100
ρ	0.01, 0.05, 0.1
N_{ni}^{max}	0.25, 0.5, 0.75, 1

Table 8: Effect of α, β, N_{ni}^{max}, ρ, and `max_ants` on the average gap

α	% gap	β^*	% gap	N_{ni}^{max}	% gap	ρ	% gap	`max_ants`	% gap
1	7.5	0.01	10.5	0.25	5.7	0.01	5.2	10	5.7
3	6.0	0.03	5.2	0.50	5.8	0.05	5.9	25	5.5
6	4.9	0.05	3.7	0.75	5.8	0.10	6.3	50	5.7
9	4.8	0.1	3.8	1.00	5.9			75	5.9
								100	6.2
Total	5.8	Total	5.8	Total	5.8	Total	5.8	Total	5.8

These results for the ACS indicate that variations in ρ, `max_ants` and N_{ni}^{max} have little impact on the performance. In contrast, changing the values of α and β^* does have a significant effect on the performance of the ACS. As explained in Section 4.3 the absolute values of α and β^* determine the weight given to the random number in the end of the construction procedure. Higher values of α and β^* lead to less randomness in the choice of the next vertex to add to the solution. Based on Table 8, we conclude that higher values of both α and β^* are beneficial for the performance, i.e., that randomness in constructing a solution should be kept low. This effect is visually shown in Figure 7, where the dark blue area represents a large optimal zone corresponding to an average gap of 1%. This figure, made using the commercial software Origin Pro, represents a 3D surface plot for a wide range of orthogonal combinations of α and β^*. In general we can conclude that the ACS solution procedure is rather robust and small changes in the parameters do not lead to huge gaps in performance.

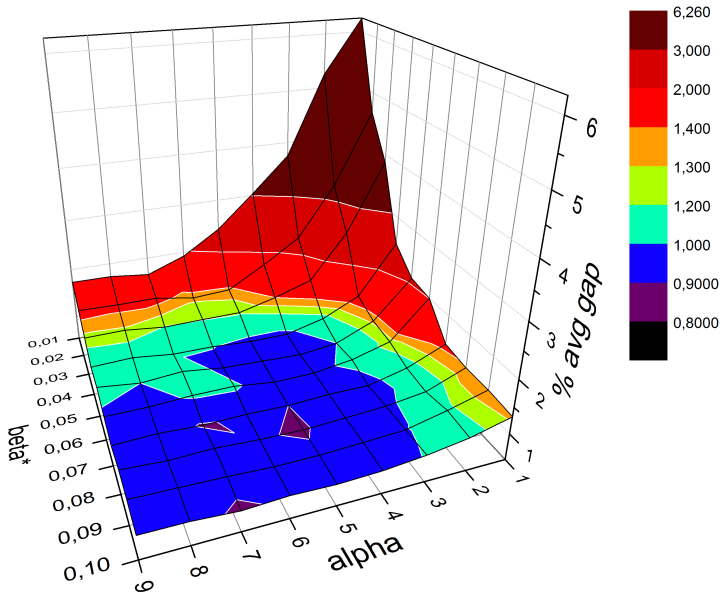


Figure 7: 3D graph showing the impact of α and β^* on the average gap, the other parameters are set at their default value

In addition to the effect on the average gap, the CPU time also remained the same when changing the values of the input parameters.

Table 9 yields insights into the performance of some design decisions. The effect is measured by leaving the component out of the algorithm and executing the altered algorithm on the set of 21 instances. For each alternative design of the algorithm, the average and maximal gap as well as the computation time is indicated.

Table 9: Effect of the design components on average and maximum gap

Metric	Normal	No 2-Opt	No insert	No 2-Opt & No insert
avg gap	1.4%	2.5%	6.9%	8.4%
max gap	6.1%	12.3%	30.9%	30.3%
avg CPU	0.5	0.5	0.3	0.2

The backbone of the ACS is formed by the ant construction procedure together with the pheromone memory effect. It can be noted that the effect of 2-Opt is small but significant, especially to reduce the maximum gap. Executing more 2-Opt procedures per iterations turns out to have a drastically lower marginal benefit. The added value of the 2-Opt procedure lies in the extra diversity that is created by searching for an alternative sequence of the included vertices.

6. Conclusions

This paper presents the time-dependent orienteering problem, an extension to the orienteering problem in which the travel time between two vertices depends on the departure time at the first vertex. In the considered tourist and logistical applications, this modification models multi-modal transport functionalities, as well as congestion troubled vehicle routing planning.

Apart from a practical mathematical formulation for this problem that obtains optimal solutions for small problem instances, the main contribution of this paper is a fast local search based metaheuristic, inspired by an ant colony system. The local search mechanism itself is a time-dependent insertion procedure, sped up by a local evaluation

metric. Moreover, realistic time-dependent test instances with known optimal solution are developed based on the original time-*independent* OP instances in combination with a well performing speed model for TD-VRP, a closely related problem.

The algorithm obtains high-quality results on these instances requiring very small computation times, even for instances with around 100 nodes. An average run obtains solutions with a score gap of only 1.4% using 0.5 seconds of computation time. For 44% of the test instances, the known optimal solution is found. A sensitivity analysis demonstrates that the performance of the algorithm is not sensitive to small changes in the parameter settings. This indicates that robust behavior might therefore be expected when incorporating this algorithm into real-life applications. The fast execution time of this algorithm enables some interesting business applications where it is necessary to update routes when new traffic information becomes available and to provide proper guidance to drivers/tourists on the road.

Further research could focus on developing exact solution methods that are able to solve small time-dependent OPs within a reasonable amount of time. Advanced cuts will need to be developed and applied in order to successfully complete this line of research.

Obviously, the performance of our metaheuristic depends crucially on the problem-specific operators that we have developed, and that are specifically geared towards solving the TD-OP (construction procedure, insert move, and memory structure). The specific framework within which these operators have been embedded (ACS), on the other hand, carries much less weight and could most likely be replaced by another suitable metaheuristic framework without severely affecting the performance of the overall method. More specifically, since Tabu Search appears to be a successful framework to deal with the TD-VRP [19, 43, 30], it might be worthwhile to consider it as a framework to deal with the TD-OP as well. Furthermore, solving realistic extensions of the TD-OP also seems an interesting research opportunity. More specifically, the time-dependent variant of the team orienteering problem (TD-TOP) is very interesting as it allows to optimize the routing of a fleet of vehicles, instead of one vehicle only, which is certainly useful for logistic companies. Adding time windows to this problem formulation will add to the practical relevance of the developed solution methods since opening hours are very common in most practical situations.

Acknowledgments

This research was partially funded by the Agency for Innovation by Science and Technology in Flanders (IWT)

References

[1] Abbaspour, A., & Samadzadegan, F. (2011). Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications*, *38*, 12439–12452.

[2] de Aragão, M. V. S. P., Barboza, F. H., & de Freitas, V. E. U. (2010). Team orienteering problem: Formulations and branch-cut and price. *Monografias em Ciência da Computação*, *13/10*.

[3] Arkin, E., Mitchell, J., & Narasimhan, G. (1998). Resource-constrained geometric network optimisation. In *Proceedings of the 14th ACM Symposium on Computational Geometry* (pp. 307–316). New York: ACM.

[4] Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). Applying the ant system to the vehicle routing problem. In *Meta-Heuristics: Advances and Trends in Local Search Paradigms for optimization* (pp. 285–296).

[5] Butt, S., & Cavalier, T. (1994). A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research*, *21*, 101–111.

[6] Chao, I., Golden, B., & Wasil, E. (1996). Theory and methodology a fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, *88*, 475–489.

[7] Chen, H., Hsueh, C., & Chang, M. (2006). The real-time time-dependent vehicle routing problem. *Transportation Research Part E*, *42*, 383–408.

[8] Dabia, S., Ropke, S., van Woensel, T., & De Kok, T. (2013). Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, *47(3)*, 380–396.

[9] Donati, A., Montemanni, R., Casagrande, N., Rizzoli, A., & Gambardella, L. (2008). Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*, *185*, 1174–1191.

[10] Figliozzi, M. A. (2012). The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review*, *48*, 616 – 636.

[11] Fischetti, M., Salazar, J., & Toth, P. (1998). Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, *10*, 133–148.

[12] Fleischmann, B., Gietz, M., & Gnutzmann, S. (2004). Time-varying travel times in vehicle routing. *Transportation Science*, *38*, 160–173.

[13] Fomin, F., & Lingas, A. (2002). Approximation algorithms for time-dependent orienteering. *Information Processing Letters*, *83*, 57–62.

[14] Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., & Linaza, M. (2013). Integrating Public Transportation in Personalised Electronic Tourist Guides. *Computers & Operations Research*, *40*, 758–774.

[15] Gendreau, M., Laporte, G., & Semet, F. (1998). A branch-and-cut algorithm for the undirected selective travelling salesman problem. *Networks*, *32*, 263–273.

[16] Gendreau, M., Laporte, G., & Semet, F. (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, *106 (2-3)*, 539–545.

[17] Golden, B., Levy, L., & Vohra, R. (1987). The orienteering problem. *Naval Research Logistics*, *34*, 307–318.

[18] Haghani, A., & Jung, S. (2005). A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research*, *32*, 2959–2986.

[19] Ichoua, S., Gendreau, M., & Potvin, J. Y. (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, *144*, 379–396.

[20] Ilhan, T., Iravani, S. M. R., & Daskin, M. S. (2008). The orienteering problem with stochastic profits. *IIE Transactions*, *40*, 406–421.

[21] Kantor, M., & Rosenwein, M. (1992). The orienteering problem with time windows. *Journal of Operational Research Society*, *43 (6)*, 629–635.

[22] Kataoka, S., & Morito, S. (1988). An algorithm for the single constraint maximum collection problem. *Journal of the Operational Research Society of Japan*, *31*, 515–530.

[23] Ke, L., Archetti, C., & Feng, Z. (2008). Ants can solve the team orienteering problem. *Computers and Industrial Engineering*, *54*, 648–665.

[24] Kok, A., Hans, E., Schutten, J., & Zijm, W. (2010). Vehicle routing with traffic congestion and drivers' driving and working rules.

[25] Kok, A. L., Hans, E. W., & Schutten, J. M. J. (2012). Vehicle routing under time-dependent travel times: The impact of congestion avoidance. *Computers & Operations Research*, *39 (5)*, 910–918.

[26] Kritzinger, S., Doerner, K., Hart, R. F., Kiechle, G., Stadler, H., & Manohar, S. (2012). Using traffic information for time-dependent vehicle routing. *The Seventh International Conference on City logistics*, *39*, 217–229.

[27] Kritzinger, S., Tricoire, F., Doerner, K., & Hartl, R. (2011). Variable neighborhood search for the time-dependent vehicle routing problem with soft time windows. In C. Coello (Ed.), *Learning and Intelligent Optimization* (pp. 61–75). Springer Berlin Heidelberg volume 6683 of *Lecture Notes in Computer Science*.

[28] Laporte, G., & Martello, S. (1990). The selective travelling salesman problem. *Discrete applied Mathematics*, *26*, 193–207.

[29] Lecluyse, C., Sörensen, K., & Peremans, H. (2013). A network-consistent time-dependent travel time layer for routing optimization problems. *European Journal of Operational Research*, *226*, 395–413.

[30] Lecluyse, C., Van Woensel, T., & Peremans, H. (2009). Vehicle routing with stochastic time-dependent travel times. *4OR: Quarterly journal of Operational Research*, *7*, 363–377.

[31] Li, J. (2011). Model and algorithm for time-dependent team orienteering problem. In S. Lin, & X. Huang (Eds.), *Advanced Research on Computer Education, Simulation and Modeling* (pp. 1–7). Springer Berlin Heidelberg volume 175 of *Communications in Computer and Information Science*.

[32] Li, J., Wu, Q., Li, X., & Zhu, D. (2010). Study on the time-dependent orienteering problem. In *International Conference on E-Product E-Service and E-Entertainment (ICEEE)*.

[33] Malandraki, C. (1989). *Time dependent vehicle routing problems: formulations, solution algorithms and computations experiments.*. Ph.D. thesis Northwestern University, Evanston, USA.

[34] Malandraki, C., & Daskin, M. (1992). Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. *Transportation Science*, *26*, 185–200.

[35] Potvin, J., Xu, Y., & Benyahia, I. (2006). Vehicle routing and scheduling with dynamic travel times. *Computers & Operations research*, *33*, 1129–1137.

[36] Ramesh, R., & Brown, K. (1991). An efficient four-phase heuristic for the generalized orienteering problem. *Computers & Operations Research*, *18*, 151–165.

[37] Schilde, M., Doerner, K., Hartl, R., & Kiechle, G. (2009). Metaheuristics for the biobjective orienteering problem. *Swarm Intelligence*, *3*, 179–201.

[38] Souffriau, W., Vansteenwegen, P., Vertommen, J., Vanden Berghe, G., & Van Oudheusden, D. (2008). A personalised tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence*, *22*, 964–985.

[39] Sörensen, K., Sevaux, M., & Schittekat, P. (2008). Adaptive, self-adaptive and multi-level metaheuristics. chapter "Multiple neighbourhood search" in commercial VRP packages: evolving towards self-adaptive methods. (pp. 239–253). London: Springer volume 136 of *Lecture Notes in Economics and Mathematical Systems*.

[40] Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*. Hoboken, New Jersey: John Wiley & Sons, Inc.

[41] Thomadsen, T., & Stidsen, T. (2003). *The Quadratic Selective Travelling Salesman Problem*. Technical Report Informatics and Mathematical Modelling, Technical University of Denmark, DTU Richard Petersens Plads, Building 305, DK-2800 Kgs. Lyngby.

[42] Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, *35 (9)*, 797–809.

[43] Van Woensel, T., Kerbache, L., Peremans, H., & Vandaele, N. (2008). Vehicle routing with dynamic travel times: A queueing approach. *European Journal of Operational Research*, *186*, 990–1007.

[44] Vansteenwegen, P. (2008). *Planning in tourism and public transportations - attraction selection by means of a personalised electronic tourist guide and train transfer scheduling*. Ph.D. thesis Katholieke Universiteit Leuven, Centre for Industrial Management, Belgium.

[45] Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: a survey. *European Journal of Operational Research*, *209*, 1–10.

[46] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., & Van Oudheusden, D. (2011). The City Trip Planner: A Tourist Expert System. *Expert Systems with Applications*, *38*, 6540–6546.

[47] Wang, X., Golden, B., & Wasil, E. (2008). *Using a genetic algorithm to solve the generalized orienteering problem*. Springer US.